Using Apache Airavata and EasyGateway for the creation of complex science gateway front-end

Antonella Galizia[a,*], Luca Roverelli[a], Gabriele Zereik[a], Emanuele Danovaro[a], Andrea Clematis[a], Daniele D'Agostino[a]

[a] *CNR-Institute of Appled Mathematics and Information Technologies "E. Magenes", via De Marini 6, 16149 Genova, Italy*
{*galizia,roverelli,zereik,danovaro,clematis,dagostino*}*@ge.imati.cnr.it*

Abstract

The development of community-specific user interfaces of a science gateway can be a challenging task for non-IT experts. This contribution proposes an original, easy-to-use solution to tackle this issue based on EasyGateway. EasyGateway is a modern, lightweight solution for the development of science gateway able to interplay with most toolkits. In this paper we present how EasyGateway can "dress" Apache Airavata to manage experiment configurations and job submissions. We discuss the proposed approach considering a real case application represented by the Weather Research and Forecasting Model (WRF), a community model exploited by a large number of users in meteorology and climatology domains. The combined use of EasyGateway and Apache Airavata leads to an improved user experience, enhancing the model configuration phase with support in finding inconsistencies immediately, but also exploiting the possibility of accessing a potentially large set of computational resources to perform model execution.

Keywords: Science Gateways; Apache Airavata; EasyGateway; WRF

1. Introduction

Science gateways represent ecosystems of services, applications, and data for supporting activities in many scientific, engineering and education fields. Every community involved in the development of a science gateway presents different requirements due to the software and/or data it shares and the goals it aims to achieve [1].

The needs and opinions of these communities were investigated in 2014 with a survey having nearly 5,000 respondents [2]. The most relevant result, for the aim of the present paper, is that at least 40% of the participants indicated that some help by a service provider might be needed in adapting/choosing technologies and usability services.

Most of the available toolkits and software frameworks for science-gateway design decouple front-end and back-end with an API-based interface. With this approach, the gateway communities can focus their effort in the designing of the front-end solution [3, 4]. This leads to a "Platform-as-a-Service" scenario, where a separation of the community-specific presentation layer from the general-purpose middleware and "fabric" layers represents an effective solution to assure the sustainability of existing and new projects [5, 6]. An example is the Science Gateway Platform as a Service[1] (SciGaP) initiative.

*Corresponding author

[1] SciGaP, https://scigap.org

However, also the task of developing domain-specific User Interfaces (UI) may be challenging for non-IT communities, and a wrong selection of the front-end technology, combined with frequent developer turnover, can represent a major issue for the gateway sustainability [2, 7, 8]. In the following, we present how this problem can be effectively tackled by EasyGateway, a science gateway toolkit able to interplay with the major scientific platforms, as gUSE [3] in Europe and Apache Airavata [9] in the US.

EasyGateway is based on PortalTS, a set of core Web portal modules which was originally developed for the refactoring of the DRIHM portal, a science gateway for the hydro-meteorological community exploiting in its first release the gUSE framework [10]. Working on PortalTS we realized that its modular and extensible architecture represents an effective and general-purpose solution for web portal and science gateways. For this reason, we developed further modules providing basic science gateways functionalities, and the automatic generation of user friendly configuration interfaces for scientific models. The result of this process has been called EasyGateway.

The focus of this paper is to present the EasyGateway architecture, which allows us to dress existing platforms such as Airavata. This approach has the major advantage of providing a user friendly front-end toolkit for gateway development. A summary of these topics was outlined in [11]. Here the technological aspects are presented in detail by analyzing a real case of interplay between EasyGateway and Airavata to support the execution of the Weather Research and Forecasting Model (WRF) in a science gateway. The adoption of EasyGateway as a front-end for the development of a science gateway, coupled with Apache Airavata, improves the user experience with a twofold goal: the enhancement of the model configuration phase with support to avoid possible errors, and the possibility to access and exploit a potentially large set of computational resources to execute models. Moreover, thanks to the architectural design of EasyGateway, the entire system can be configured and personalized with very little effort, saving time and money, while leaving the creation of the model configuration interface to domain-expert users.

The paper is organized as follow: in the next Section related works are outlined; in Section 3 we introduce PortalTS and EasyGateway; the technological aspects for the exploitation of Airavata services are presented in details in Section 4; the analysis of the advantages derived from the combined use of the two tools are discussed in Section 5 by describing the reference user communities, the implementation of the WRF interface, the configuration and the execution of the WRF model; Section 6 concludes and presents the future research directions.

2. Related Work

Creating a science gateway requires a wide spectrum of skills and technologies, ranging from low-level interaction with heterogeneous computing resources (e.g. supercomputers, clusters, Grid and Cloud infrastructures) to high-level workflow and data management systems. All the complexity should be hidden behind a user-friendly UI.

In the context of the Distributed Research Infrastructure for Hydro-Meteorology (DRIHM) project, we adopted gUSE [12], a mature and feature-rich framework, for the design of the science gateway. gUSE extends Liferay by adding connectors to several kind of computing resources, user credential management and a workflow manager. The UI is portlet-based, i.e. pluggable user interface software components managed and displayed in web portal containers as Liferay, thus modular by design. Therefore it supports the creation of science gateways with three degrees of customization: the exploitation of the gUSE general purpose UI, with simple configuration of existing modules; the customization of the general purpose UI, e.g. for better supporting the user in experiment configuration; the deep customization of the UI and of the underlying service, by connecting to the low-level APIs [13]. Unfortunately deep customization of the UI, as required in our project, was problematic [10]. For this reason we extended our analysis to other toolkits for the development of a science gateway UI.

In [14] the authors propose the adoption of excellent web front-end technologies, namely Twitter Bootstrap and AngularJS, for the development of a science gateway front-end. Such tools are widely

adopted (Bootstrap and AngularJS are used in 13.2% and 8.9% respectively of top 10k websites[2]). While AngularJS is now phasing out [15], it remains a solid tool with a large ecosystem. The main downside of this approach is the lack of high-level widget and features designed for science gateways, leaving all the effort to the gateway developers.

Rapid [16] has been designed to speedup the development of portlets. Its main aim is to reduce the development time of the UI and to promote an easy way to share and maintain these portlets by domain specialist themselves. Rapid can complement science gateways framework based on the portlet technology as gUSE.

The HUBzero platform [17] is an open source platform for the creation of Cloud-based science gateways. The main goal is to share legacy processing tools, mostly presenting their own UI, among community members. HUBzero relies on the Rappture toolkit [18] for wrapping the legacy UI through the web browser and on the resources provided by the Extreme Science and Engineering Digital Environment (XSEDE) for job execution. The derived hubs have been able to grow to tens of shared tools (i.e. nanoHUB), actually HUBzero excels in supporting users willing to share a processing tool. It is also available on the AWS marketplace, to further simplify the creation of a new HUB. However, tools configuration is not web-based since UI generated by Rappture are stand-alone applications installed client-side.

GenApp [19] is a framework for building complex applications from a collection of executable modules. It wraps executables, providing a UI for parameters configuration and module composition. Each module must accept and produce JSON inputs and outputs. Control files, defining module UI and I/O interfaces are specified in JSON as well. GenApp may generate a Qt/C++ GUI or a HTML5/PHP web app. It has been extended to support local execution and submission as Airavata jobs. It is well suited for wrapping JSON-centric applications, while it fails handling large applications managing more complex data, i.e. binary datasets or large data files.

SEAGrid [20] is a production community cyberinfrastructure resource developed under the NSF Middleware Initiative. Its initial focus was in Chemistry, but presently it has evolved to encompass scientific and engineering applications. SEAGrid provides a desktop and a web browser based gateway access. The latter represents a rather simple UI, composed of basic form widgets, to interact with an Apache Airavata instance [21] for creating, submitting and managing experiments.

EasyGateway aims at providing a light yet elegant web-based UI, as in [14]. Main benefits are: availability of high level widgets, implemented as AngularJS directives, as in desktop applications generated by Rappture or GenApp; a web-based solution, relieving developers from the effort of supporting multiple platforms, managing releases and updates; complete freedom on the executable modules, without restricting to JSON I/O, which is problematic for those applications producing large binary data (i.e. WRF).

3. EasyGateway

EasyGateway is a modern, lightweight, toolkit for the development of science gateway. It has been developed as a set of additional modules for PortalTS, a web portal initially developed for the refactoring of the DRIHM portal [22]. The project originally focused on the design and the execution of forecasting chains, composed by heterogeneous Meteorological and Hydrological models, on heterogeneous computing resources [23]. In particular, significant attention has to be paid to the user experience during workflow configuration, in order to avoid consistency errors in the models composition. We realized in fact that portlets, forbidding direct intra-portlets communication, were not the proper solution for the cross-check of consistency errors. Therefore in PortalTS we clearly decoupled server-side REST services (e.g. persistence, provenance) and the experiment configuration UI, running client-side and able to support the users in consistent parameter selection.

Another issue is represented by the fact that gUSE is composed by tightly coupled modules, a re-deploy may potentially require a long downtime. Updates

[2] https://trends.builtwith.com

are performed on a best-effort policy and often are not backward compatible, thus requiring demanding update on the community-developed software components.

In the following we discuss PortalTS architecture and EasyGateway evolution.

3.1. PortalTS

PortalTS[3] is a web Portal developed in Typescript using the NodeJS and Express frameworks, backended with MongoDB. It is composed by reusable modules and implements standard features available for a web site (e.g. user management and registration) along with other valuable characteristics (such as a simple API for data persistence) that enable fast development of custom modules. The PortalTS architecture is depicted in Figure 1.

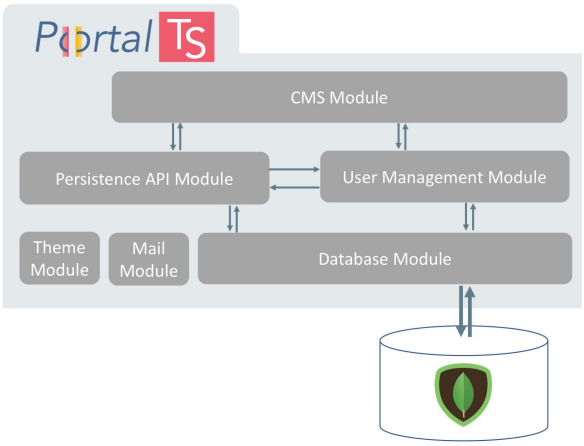

Figure 1: The PortalTS architecture.

First, let us define a module as a component that implements and exposes a feature, and can also use features exposed by other modules. It is a very general component representing, for example, a set of web pages, a web service, a web app (aka Single Page Application), a set of static files like css files, images, or something different. According to the NodeJS philosophy, each module should be as simple as possible and implement a single functionality.

PortalTS loads modules on the bootstrap phase, using a configuration file to specify module loading order. Since PortalTS and any module are developed in Typescript, module error loading caused by typos and other typical JavaScript errors (e.g. undefined functions or undefined function arguments) is dramatically reduced, resulting in an improved software reliability and stability. The most important modules are briefly presented in the following.

The **Database module** defines and manages the connection with the MongoDB database. Despite its simplicity, this module has a core value since it is exploited by higher-level modules for the communication with the database. MongoDB has been chosen because it is extremely well integrated with NodeJS. Moreover, there are some high level libraries, e.g. mongoose[4], that are stable and maintained, making them usable in a production environment.

The **User Management module** defines an API for a complete authentication system, including user registration, login, and administration pages. It implements also the concepts of role and group, at the basis of the authorization mechanisms for the accessibility and usability of pages, modules and other entities.

The **Persistence API module** defines the interface to store, retrieve and manage heterogeneous data on the MongoDB database. It exposes both a RESTful public and a private API, that can be directly used by other modules, as the **Content Management System (CMS) module** described in the following. The RESTful public API has the important role of allowing the storage of data directly from a web app, that can be built upon this modules. The Persistence API defines entities and collections. A collection is a set of entities, while an entity represents possibly heterogeneous data stored with some additional metadata, like creation time, update time, the owner and the authorized users. Each entity can belong to a single collection. Most common entities handled by the

[3]https://portalts.it/

[4]http://mongoosejs.com/

persistence API are model configuration namelists, input datasets, etc., while huge dataset are stored as regular files and just referred in the persistence API. Relying on MongoDB, a document-based data store, we do not pose constraints on the data structure. A detailed description on the Persistence API is available on-line[5].

The Persistence API relies on the User Management Module to ensure security and user authentication on the data. By default, an entity is only accessible by the owner, but the access policies can be changed, using a group-based policy. The Persistence API Module is fundamental since it allows to store and retrieve persisted data without any effort, enabling a ready-to-use persistence layer. Moreover, this layer is integrated with an AngularJS library that implements all methods necessary to give a quicker and simpler access to the persistent data.

The CMS module defines web pages for user login and registration, and it allows the creation of user-defined web pages and menus. Each web page or menu element can be publicly available or accessible by a particular group of users, since the CMS module uses the Persistence API module. Images and files can be managed by using the Repository module, and then exploited by the web pages. There are also some further basic modules, like the Theme module, that defines the web pages header and the footer to define a standard look and feel of a portal instance, and the Logging module for storing the requests received by all modules, together with possible errors and exceptions.

Although PortalTS is a very young project (started in January 2016), it has been already used as the base for the development of the science gateway of the EU-funded FP7 Exploring the X-ray Transient and variable Sky (EXTraS) project[6] [24]. The resulting portal enables users to perform image analysis operations on the database collected by the European Photon Imaging Camera (EPIC) onboard the ESAs X-ray space observatory XMM-Newton. EXTraS portal currently supports periodicity and transient analyses [25]. In particular, the portal provides users with the possibility to create, submit and manage the different analysis experiments based on software developed within the EXTraS project. This module is based on AngularJS and it is a complete web app, without any server side code. It uses the Persistence API to store and retrieve experiments data, and it activates the other portal modules corresponding to the different operations available.

3.2. EasyGateway modules

On the basis of this experience we decided to optimize and re-engineer the modules developed ad-hoc for the EXTraS portal. The aim was to provide them as a general-purpose science gateway toolkit based on PortalTS, named EasyGateway. Main areas of improvement have been: experiment UI definition, workflow management, computing resource interfacing. With respect to PortalTS, EasyGateway provides much more flexibility on each one of these aspects. In particular, the new modules provide workflow configuration and submission, and the automatic generation of possibly complex, yet user friendly, configuration interfaces for scientific models. In details, EasyGateway is composed by the Model Configuration module, the Workflow Configuration module and the Submission Handler module, shown in Figure 2. Each module is a Single Page Web Application leveraging the ready-to-use components and APIs exposed by PortalTS to enable users management, security and to persist users data.

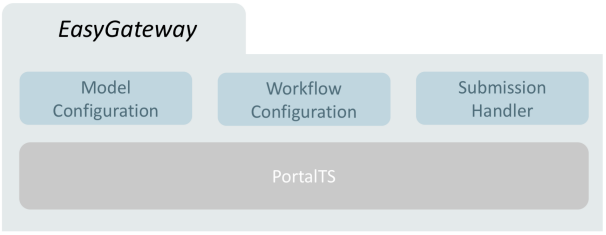

Figure 2: The EasyGateway architecture

The **Workflow Configuration module** provides users with the possibility to design, configure, submit

[5]https://portalts.it/docs/webPersistenceAPI
[6] http://portal.extras-fp7.eu

and share experiments in terms of workflows, composed of software tools provided within the science gateway. It represents the main interface and it interacts with the other modules for performing these operations. Furthermore, it allows a user to retrieve a previously saved or shared workflow, and to edit it, before execution. The Workflow Configuration module actually manages the design of a workflow and the configuration of each single module by exploiting the UI generated with the Model Configuration module. The workflow, once configured, is submitted on a selected set of resources; the monitoring of the execution is performed by providing a full view of the status results and logs of all models composing the workflow.

In order to improve re-usability and maintainability, the Workflow Configuration module delegates submission and monitoring of jobs to the **Submission Handler module**, that has been conceived as an independent component, to be customized for each class of resource, middleware layer or job submission system that the gateway exploits for the workflow execution. It uses the typical decorator pattern, in which the behavior of the Submission Handler is extended and modified according to the underlining middleware, without modifying the exposed API.

The **Model Configuration module** is an innovative element, conceived for the automatic design and support of suitable interfaces to collect the configuration parameters of any software tool provided within the gateway. This module is available only for gateway developers because it allows to automatically generate a form based model configuration interface through a user friendly graphical interface, the Json-GUI builder[7]. Moreover, it is possible to easily define and manage model metadata, like names or types, as well as the configuration parameters with associated metadata, like field type (select, float, integer, date/time, etc.), parameter default value, and other specific properties (e.g. the possible options for a select type). These features have been implemented using Json-GUI[8], a front-end library developed as a set of reusable AngularJS directives that allows the dynamic generation of full-featured form-based web interfaces including validation and constraints. The interface is dynamically built starting from the JSON file created by the Json-GUI builder, that corresponds to the ability to modify the interface by simply changing a configuration object. Once the model interface has been defined, the model is automatically added to the set of available models in the Workflow Configuration module, and it is ready to be instantiated and configured in a workflow.

[7]https://github.com/portalTS/Json-gui-builder
[8]https://github.com/portalTS/Json-gui

4. The interplay with Airavata

Apache Airavata is a powerful middleware, supporting the development of solid and feature-rich science gateways, thanks to its support to long running applications and workflows on distributed computational resources. The joint use of EasyGateway and Apache Airavata leads to a rich user interface, with support for submission on a large set of middleware and queue managers. In this Section we firstly discuss the motivations and the beneficial points of the work, and then we provide a complete description about the strategy implemented to actually connect EasyGateway and Apache Airavata.

4.1. Motivations

The interplay of EasyGateway and Apache Airavata provides mutually beneficial elements that improve the features of both the individual systems. Starting from the EasyGateway point of view, this approach brings the ability to inherit exploitation of a large number of computational resources and distributed computing infrastructures. On the basis of our previous experiences, this represents an added value when developing science gateways aimed mainly at the execution of multidisciplinary applications.

Moving to the Apache Airavata point of view, this approach offers an enhanced graphical environment that simplifies the configuration of a job. In fact, we provide tools for handling a large set of parameters, selection of custom parameters is supported by user-defined AngularJS directives, consistency check of the

Table 1: The mapping implemented between the Apache Airavata and the EasyGateway Data Models

Apache Airavata	EasyGateway
Project	The same project for any experiment
Experiment	Each workflow is mapped on an Experiment
Application	Each Job of a workflow is mapped on an Application

parameters is performed on the fly and javascript functions handle the generation of configuration files required for jobs submission. Moreover, we exploit EasyGateway for the management of authorization and sharing policies.

Starting from the latter, EasyGateway provides basic functionalities for the management of users and roles, along with a rich administrative interface; this automatically provides Airavata with authentication and, as a consequence, authorization handling. Thus, it is straightforward for the administrator to grant accessibility rights to single portal features, like permitting only to specialized users to define application interfaces. In this way, EasyGateway filters the users also at the single application granularity, deciding which user groups will be allowed to configure and execute each application. Moreover, users have the ability to limit/share their products: when a specialized user defines an application interface, she/he has the ability to restrict its accessibility. Users can easily share their experiment configurations and results with different groups or make them public. Subsequently, users can analyze the results of shared experiments, without the need to execute again an application with the same configuration or to use external tools; moreover, they can derive their own configuration, varying parameter values.

The second added value is represented by the automatic definition of enhanced application interfaces. While Apache Airavata has basic features for providing application inputs, exploiting EasyGateway, a specialized user can design an interface where the input parameters: a) can be validated before experiment submission; b) can be processed, to produce complex configuration files at runtime. Beside the basic parameter types, there is the possibility to define some more complex types: domains, datetime and fileupload. The first allows to define geo-referenced points and rectangular regions over a geographical map; the datetime permits the definition of date and time parameters, while the fileupload gives the user the ability to upload one or more files. For each parameter, the module allows one to define different constraints with different error messages, where each constraint can be constituted by one or more conditions. Moreover, within each condition, the user can compare the value of the parameter with the value of another one, and/or with a static value. It is also possible to define an help text, in order to show to the user a hint for the input completion.

The last added value is the ability to create configuration files. Actually, when a GUI is used to configure a model/job, it may be necessary to transform the data inserted in the form into configuration files that will be used for the run. The Workflow Configuration Module supports, by default, three different file formats: a classical key-value format, JSON and XML. It is also possible to define a custom function for each field, that emits a transformed value of the parameter. This feature is extremely useful when a custom format is required. For example, the value of a datetime parameter must be formatted in a specific standard. Also the geographical domain may require a specific projection into a different coordinate system, as exemplified in Section 5. Finally, beside the three standard file formats, it is possible to define a custom function that generates the final configuration file. In some case, this functionality is necessary, since the configuration file expected by the model can be very complex and different from the supported standards. For example, the WRF model expected a modified key-value configuration file, in which, for some of the keys, the value shall be repeated for the number of geographical domains drawn by the user. This point is presented in the next Subsection.

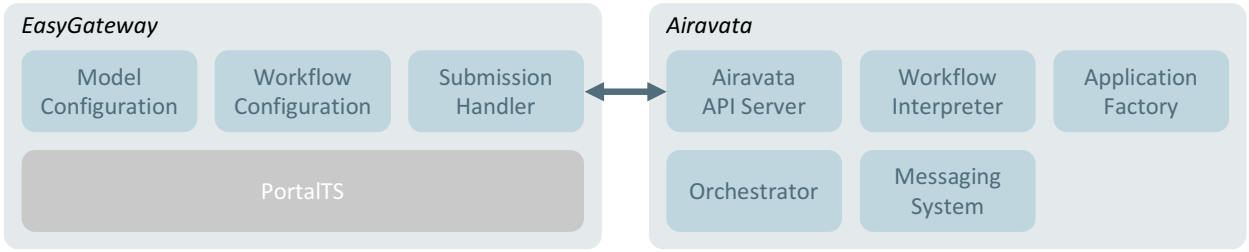

Figure 3: The EasyGateway interaction with Apache Airavata.

4.2. Enabling application submissions

The exploitation of Apache Airavata services requires to investigate four aspects: 1) the mapping between the EasyGateway and Airvata data models; 2) the creation of a two-way communication channel exploiting the available APIs exposed by Airvata; 3) the communication of the input configuration files, generated by the EasyGateway Workflow Configuration module, to Apache Airavata; 4) the user management and data sharing.

As for the first point, Airavata implements a data model defined by several entities: Project, User, Group, Gateway and Experiment. We mainly focused on the Experiment data model that nicely matches EasyGateway workflow, jobs and job descriptions. In particular, as depicted in Table 1, a unique Airavata Project is used to contain the workflows of all users. In our work, we decided to limit workflow size to just one job, in order to directly use Apache Airavata without the necessity of a workflow manager. Each workflow is mapped to an Airavata Experiment that is dynamically created. Finally, an Airavata Application is used to map each Job.

As for the second point, we designed the customization of the Airavata Submission Handler, able to exploit Apache Thrift-based API exposed by Apache Airavata API Server. We use the Java implementation of the API, wrapping the submission and monitoring of a job in a command-line Java application. The Java application allows us to test the Apache Airavata API independently from EasyGateway, and the application is easily integrated in the Submission Handler developed in Typescript. A more effective implementation could be the direct integration of the generated Typescript (or JavaScript) API stubs. However, at the moment, the Apache Airavata repository contains scripts for the generation of the API only in Java, Python and CPP. Use of the Java API is straightforward. Starting from an applicationID, it is possible to generate lists of inputs and outputs needed. When the input list has been completed with the necessary parameters, it is possible to setup a new experiment using the applicationID and the input list. Once the experiment has been created, it is also necessary to create a new resource scheduling model, which shall contain the ID of the chosen computational resource, and also the common scheduling parameters, such as number of CPUs, maximum memory, wall time and so on. Finally, it is possible to perform the launch of the configured experiment that, if everything is correct, will return a new unique experiment ID, that can be used to query Airavata in order to check the experiment status.

As for the third point, the Submission Handler Module retrieves the generated model configuration files needed for execution of the experiment, and communicates files to Apache Airavata by exploiting the input list defined in the API and exposed in the Airavata API Server (Figure 3). In Airavata the user can select the computing resource, among those granted to a project that provide a specific application. In our test case we preferred to support the users in this phase by implementing a simple scheduling policy. Nevertheless, it is absolutely possible to let users make this choice through EasyGateway.

With respect to the last point, user identity management and resource sharing are available either in Airavata and in EasyGateway; so we had to chose

which service would have been responsible for user management and resource sharing. The two options were: Airavata services and data model behind EasyGateway UI; or EasyGateway services for user management and data sharing and Airavata services for job submission (with a community account). The first approach, while providing a deeper integration, required a major re-write of EasyGateway, so we decided to use the EasyGateway facilities.

5. Testing the interplay with WRF

5.1. The user communities

As already mentioned in other parts of this paper, the strategy presented is the result of in depth interactions with two different scientific user communities that provided the initial set of requirements and suggested progressive refinements and improvements both for the user interface and for the set of required functionalities. The two communities belong two different domains and are a group of Hydro-Meteorological (HM) scientists and a group of Astronomy and Astrophysics (AA) scientists.

The first experience was driven by the requirements of the HM community. Specific requirements of the community were the use of distributed resources for the execution of workflows combining meteorological and hydrological models. The process started with an extensive requirements collection [26] led to the use of a previous version of the portal widely used during the DRIHM project lifetime, and to successive modifications to address specific needs derived from further co-operations across the Atlantic Ocean to study specific High Impact Weather Events [27].

The tools developed for the HM community were adapted to the case of AA scientists. Here the most important requirements are represented by an effective support to the continuous improvement of the software, together with the possibility to share experiment configurations and results. With respect to the HM case, the software modules to perform the experiments relies on common libraries and toolkits but are developed (and continuously updated) by small research groups. The adoption of EasyGateway resulted in a fast deployment of these modules and corresponding UI.

There are a few dozen users registered in the EXTraS portal, having only been released and presented to the AA scientific community in June 2017 [28]. The HMR community is composed of two hundred users. Apart from the scientific communities, PortalTS has been adopted also in commercial products and the most active commercial portal based on it has several thousand active users.

5.2. The WRF case

To validate our approach and evaluate the actual benefits of the integration, we ran a real case application in the integrated environment. Specifically, we consider the Weather Research and Forecasting (WRF) model[9], a numerical weather prediction and atmospheric simulation system. It represents the reference model for a large community of users, and can be defined as a flexible, state-of-the-art, portable code with high computational requirements. Many researchers investigated its use on parallel and distributed architecture [30], [31].

WRF is very complex, with different inputs and input-types, an extremely large set of parameters and not-trivial dependencies among such parameters. In fact, a set of parameters not carefully tuned easily results in unstable numerical predictions. WRF Portal [10] offers a rich interface for WRF configuration, supporting geographic domain definition and several map projections. Unfortunately, it cannot support the user in enforcing a coherent configuration. Moreover it is distributed as a desktop application that requires installation and configuration, and job submission is barely supported.

Since the configuration of WRF is so error-prone, we have developed a fully-featured web-based user interface for the WRF model using EasyGateway, and this process is described in the following. To enable the model execution in potentially complex environment, we exploited Apache Airavata to hide the complexity of these operations from the user. By adopting a web-based approach, the user is fully relieved from the software installation duties.

[9] http://www.wrf-model.org
[10] https://esrl.noaa.gov/gsd/wrfportal/

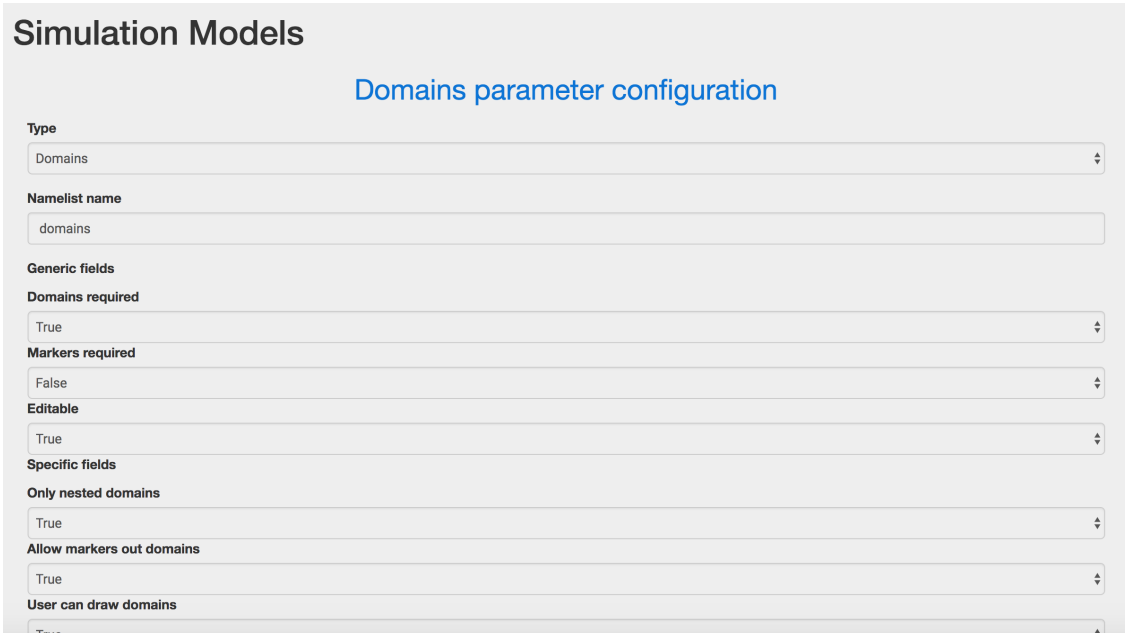

Figure 4: The UI for define parameters

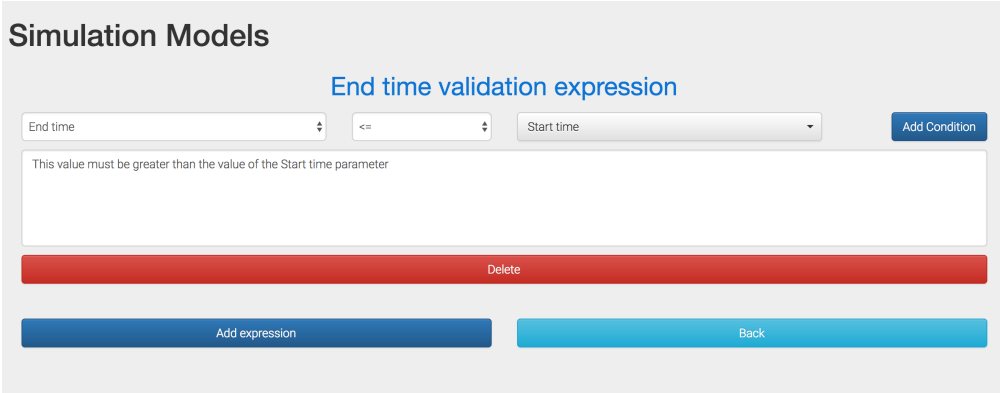

Figure 5: The UI to add constraint in the definition of a model.

5.2.1. Defining a GUI

Based on the experience gained through a long collaboration with domain experts, we designed the interface for the WRF model, enriching the one presented in [29]. We group parameters in different sections; for each section, the included parameters have the same topic. We decided to prioritize the most intuitive sections, positioning the most specific ones in the bottom of the interface. The six sections are: Domains, Time Control, Run Options, Physics Options, Diffusion and Dynamics Options and Submission Options. For each parameter it is possible to define the general behavior: a) whether the parameter is mandatory or not; b) the possible default

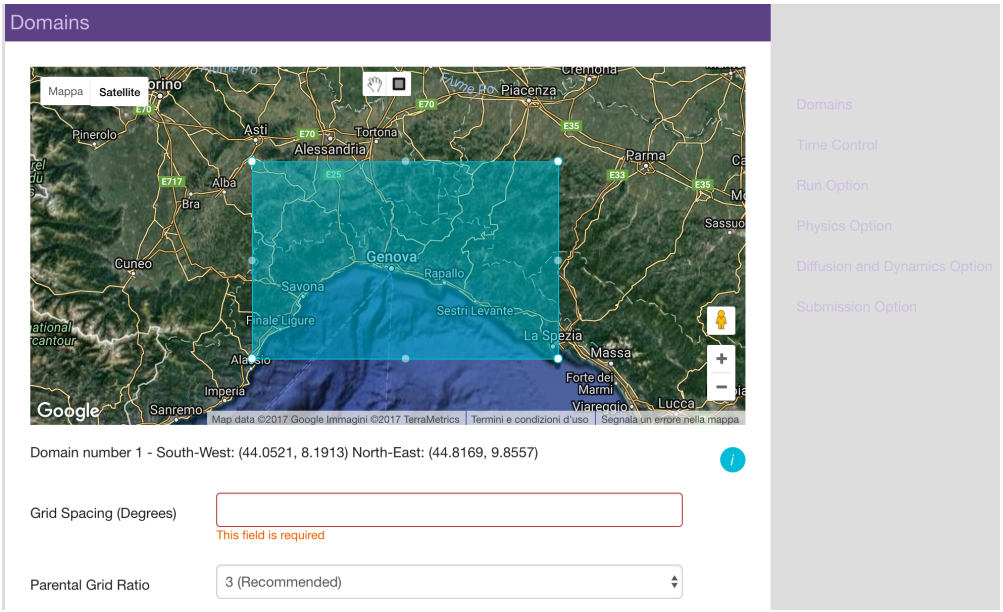

Figure 6: The UI to add constraint in the definition of a model.

value; c) whether the parameter value can be edited or not. In addition, for the datetime parameter type, it is also possible to specify whether the parameter is composed of the date, the time or both. An example of parameter definition for the Domains section is depicted in Figure 4. Please note that the first parameter is a type Domains; this corresponds to integration of the Google Map JavaScript library, enhanced with the possibility for the user to draw up to three rectangles, each one representing a geographical domain. Besides defining the parameters composing the configuration interface, their types and their values, the Model Configuration module allows the user to define some custom validations for the single parameter as well as for the dependences among them. In the formalization of the WRF model interface, we used this feature more than once: in the Time Control section, for example, we defined a constraint to prevent a user from setting the Start Date parameter older than the End Date parameter. Moreover, in the Domain section, we would not allow a user to draw siblings domains; thus, we defined a constraint to only permit the drawing of nested domains.

These constraints can be defined very easily using the Model Definition Module, as depicted in Figure 5.

5.2.2. Configuring the model through the GUI

Due to the design adopted for the interface, the user is guided and supported in filling in the different sections, with immediate feedback in case of an error or inconsistencies in the various parameters. This real-time feedback leads to an easier and less error-prone interface, speeding up the setup time also for very complex run configuration, and potentially saving computational resources.

As already mentioned, a very valuable aspect of the interface is the ability to define the geographical domain using the modern Google Map-based interface. A custom function has been defined to automatically convert the Google Map domain, which uses the Mercator projection, into Rotated Lat-Long projection required by domain experts. This step relieves the user from performing complex and error-prone calculations to correctly set up the WRF configuration file. The corresponding result is depicted in Figure

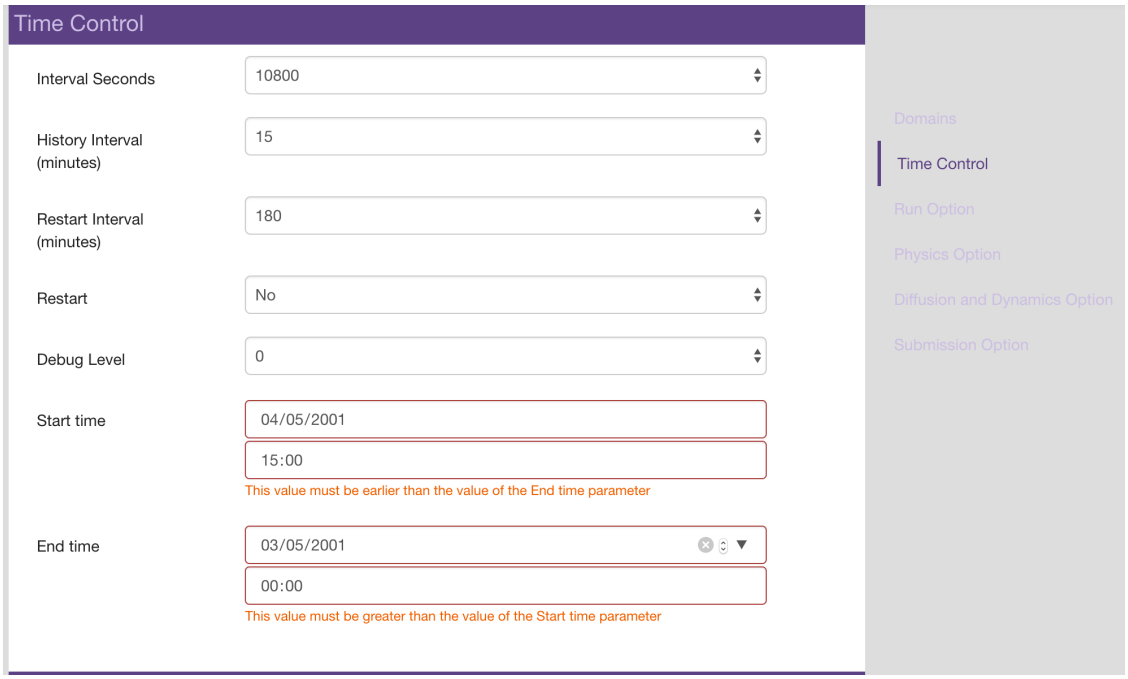

Figure 7: The UI to add constraint in the definition of a model.

6.

After domain selection, the user can move to the following panels, and select the simulated time frame (and related parameters), runtime parameters (restart info, debug level) and the available physics options. The last step is the selection of the diffusion and dynamics options. Once the user has entered a short description of the experiment, she/he can save the configuration for a later execution or review. If the consistency step is passed, the simulation can be submitted for execution on the most appropriate computational resource.

Our community of WRF users identified a set of nearly 20 parameters, which are the most relevant for event-based (simulation of a time range lasting from a few days to a couple of weeks). Further parameter tweaking can be performed on the generated namelist.

5.2.3. Submitting and Running WRF through Airavata

The installation, configuration and execution of the WRF model can be time-consuming and out of the expertise of non-IT scientists, since it requires the ability to access and use large parallel resources, which are typically geographical distributed and equipped with local schedulers. For these reasons, we installed the WRF model on a PBS-based cluster, and registered it in the Airavata Application Factory. In this way, we have also enabled the ability to execute the WRF model through the Airavata APIs. For testing purposes, we manually executed the model through standard Airavata web interface, using namelists manually created by expert meteorologists. The next step is the connection of the WRF application defined in the Airavata Application Factory with the EasyGateway portal.

The WRF configuration files, in the form of two different namelists, are produced by the EasyGateway UI through a custom routine specifically developed.

```
simpleExperiment = ExperimentModelUtil.createSimpleExperiment(gatewayId, projectId,
        userName, experimentName, experimentDesc, applicationId, experimentInputList);

scheduling = ExperimentModelUtil.createComputationResourceScheduling(resourceId, cpuCount,
    nodeCount, numberOfThreads, queueName, wallTimeLimit, totalPhysicalMemory);

userConfigurationData.setComputationalResourceScheduling(scheduling);
simpleExperiment.setUserConfigurationData(userConfigurationData);

experimentId = airavataClient.createExperiment(auth, gatewayId, simpleExperiment);
airavataClient.launchExperiment(auth, experimentId, gatewayId);
```

Figure 8: Airavata job submission

The namelist content is generated starting from the parameters configured by the user through the GUI. The generated namelists are then used by the Submission Handler and forwarded to the Apache Airavata submission APIs. The whole Airavata Experiment configuration and job submission is fully automated and handled by EasyGateway; in particular, the namelists are used to configure the input list during the creation and configuration of the Apache Airava experiment that is submitted, providing application specific parameters (geographical domain, timestep, etc.). Other user-defined parameters contributes to the definition of the Airavata experiment, namely the scheduling specific parameters like number of cores/nodes, amount of memory, etc. The time required for namelist generation and job submission is usually a fraction of a second.

The actual code performing the experiment definition and job submission is a short snippet (Figure 8). The Job status is then monitored by the Submission Handler, by querying the Airavata APIs, and notified to the user is near-real-time. In our scenario (a limited number of long-lasting jobs) EasyGateway has a negligible impact on performances, which are completely dominated by queuing and execution time. Moreover EasyGateway UI is designed to scale horizontally. The combination of an efficient UI, simple to build, yet effective in addressing complex requirements such as data cross-validation and generation of custom namelist, and a fully automated job submission managed by Airavata, has been extremely welcome from the Hydrologist and Meteorologist communities.

6. Conclusions and Future Works

One of the most promising trends in the gateway developer and user community is represented by the approach to decouple provisioning of generic middleware functionalities, common to many science gateways, from the development of the custom interfaces and features specific to each gateway reference community.

In this contribution we presented the architecture and the main features of EasyGateway, a front-end toolkit for science gateway development. EasyGateway represents a feasible solution to tackle domain-specific tasks in particular for non-IT experts, and it is able to interplay with the major scientific platforms providing general-purpose middleware and "fabric" layers. In particular we discussed how the joint use of EasyGateway and Apache Airavata represents a real coupling case showing the mutual beneficial elements. This analysis has been carried out by presenting a fully-featured user interface for the WRF model. The configuration of WRF is very complex, with different input and input-types. Therefore this process can be very error-prone, since it is very easy to set up an inconsistent run wasting CPU time due to configuration mistakes. Thanks to EasyGateway it was extremely easy to define constraints between different parameters that lead to an immediately validation of the WRF configuration without writing any line of code. The following tasks, i.e. resource allocation, job submission and monitoring, are fully automated. We are looking forward to enrich logging and monitoring services, as performed in WRF4G [32].

The present work focused on the execution of sin-

gle, even if complex, applications. The main future direction is represented by the provisioning of an enhanced accounting and resource management for operational gateways and a full support for the Apache Airavata workflows. In fact while EasyGateway is already able to support full workflow execution for other middleware and queuing systems, the workflow management in Airavata is a work in progress[11]. Nevertheless, a pre-defined workflow, as for example those considered in DRIHM, can be installed and provided through Airavata as a "single application" represented by a script orchestrating the execution of software composing it. This strategy has the advantage to avoid temporary tricky, non-portable solution. Indeed the complexity of the solution is moved to EasyGateway, that is is able to fully support workflow execution.

7. References

[1] G. Andronico, V. Ardizzone, R. Barbera, et al (2011). E-Infrastructures for e-science: a global view. Journal of Grid Computing, 9(2), 155-184.

[2] K.A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. Wernert, M. Pierce, S. Marru, S. Michael (2015). Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community. Concurrency and Computation: Practice and Experience, 27(16), 4252-4268.

[3] Kacsuk, P. (Ed.) (2014). Science Gateways for Distributed Computing Infrastructures: Development framework and exploitation by scientific user communities. Springer. ISBN 9783319112671

[4] S. Gesing, J. Krger, R. Grunzke, S. Herres-Pawlis, A. Hoffmann (2016). Using Science Gateways for Bridging the Differences between Research Infrastructures. Journal of Grid Computing, 14(4), 545-557.

[5] M. Pierce, M. Suresh; M.A. Miller, A. Majumdar, B. Demeler (2013). Science Gateway Operational Sustainability: Adopting a Platform-as-a Service Approach. Proceedings of the 11th Gateway Computing Environments Conference. Doi: https://doi.org/10.6084/m9.figshare.790760

[6] N. Wilkins-Diehr, K.A. Lawrence (2010). Opening science gateways to future success: The challenges of gateway sustainability. Proceedings of the Gateway Computing Environments Workshop (GCE 2010), 1-10. IEEE.

[7] T. Piontek, B. Bosak, M. Cinicki, P. Grabowski, P. Kopta, M. Kulczewski, et al.(2016). Development of Science Gateways Using QCGLessons Learned from the Deployment on Large Scale Distributed and HPC Infrastructures. Journal of Grid Computing, 14(4), 559-573.

[8] K.A. Lawrence, N. Wilkins-Diehr (2012). Roadmaps, not blueprints: paving the way to science gateway success. In Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond, 40. ACM.

[9] M.E. Pierce, S. Marru, L. Gunathilake, D.K. Wijeratne, R. Singh, C. Wimalasena, S. Ratnayaka, S. Pamidighantam (2015). Apache Airavata: design and directions of a science gateway framework. Concurrency and Computation: Practice and Experience, 27(16), 4282-4291.

[10] D. D'Agostino, E. Danovaro, A. Clematis, L. Roverelli, G.Zereik, A. Parodi, A. Galizia (2016). Lessons learned implementing a science gateway for hydrometeorological research. Concurrency and Computation: Practice and Experience, 28(7), 2014-2023.

[11] D. D'Agostino, L. Roverelli, G. Zereik, E. Danovaro, A. Clematis, A. Galizia (2017) Dressing Apache Airavata services with automatically user-generated interfaces. Proceedings of the 11th Gateway Computing Environments Conference. Doi: 10.6084/m9.figshare.4490006.v2

[11] goo.gl/fhWd31

[12] A. Balasko, Z. Farkas, P. Kacsuk (2013). Building science gateways by utilizing the generic WS-PGRADE/gUSE workflow system. Computer Science, 14(2), 307.

[13] T. Kiss, G. Terstyanszky, P. Borsody, P. Kacsuk, . Balasko (2014). Developing science gateways at various levels of granularity using WS-PGRADE/gUSE. In Science Gateways for Distributed Computing Infrastructures, 111-122. Springer International Publishing. ISBN 9783319112671

[14] V. Balasubramanee, C. Wimalasena, R. Singh and M. Pierce (2013). Twitter bootstrap and AngularJS: Frontend frameworks to expedite science gateway development. In Cluster Computing (CLUSTER), 2013 IEEE International Conference on (pp. 1-1). IEEE.

[15] R. Parsons, M. Fowler et al. ThoughtWorks Technology Radar vol. 16, https://www.thoughtworks.com/radar

[16] J. van Hemert, J. Koetsier, L. Torterolo, I. Porro, M. Melato, R. Barbera (2011). Generating webbased user interfaces for computational science. Concurrency and Computation: Practice and Experience, 23(3), 256-268.

[17] M. McLennan, R. Kennell. (2010). HUBzero: a platform for dissemination and collaboration in computational science and engineering. Computing in Science & Engineering, 12(2), 48-52.

[18] M. McLennan (2011). Creating and Deploying Scientific Tools. Tutorial at the HUBbub Conference.

[19] E.H. Brookes, N. Anjum, J.E. Curtis, S. Marru, R. Singh, M. Pierce (2015). The GenApp framework integrated with Airavata for managed compute resource submissions. Concurrency and Computation: Practice and Experience, 27(16), 4292-4303.

[20] S. Pamidighantama, S Nakandala, E. Abeysinghe, C Wimalasena, S. Rathnayakae, S. Marru, M. Pierce (2016). Community science exemplars in seagrid science gateway: Apache airavata based implementation of advanced infrastructure. Procedia Computer Science, 80, 1927-1939.

[21] S. Nakandala, S. Pamidighantam, S. Yodage, N. Doshi, E. Abeysinghe, C.P. Kankanamalage, S. Marru, M. Pierce (2016). Anatomy of the SEAGrid Science Gateway. In Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale (p. 40). ACM.

[22] D. DAgostino, E. Danovaro, A. Clematis, L. Roverelli, G. Zereik, A. Galizia (2016). From Lesson Learned to the Refactoring of the DRIHM Science Gateway for Hydro-meteorological Research. Journal of Grid Computing, 14(4), 575-588.

[23] D'Agostino, D., Clematis, A., Galizia, A., Quarati, A., Danovaro, E., Roverelli, L. et al. (2014). The DRIHM project: a flexible approach to integrate HPC, grid and cloud resources for hydro-meteorological research. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC14), pp. 536-546, IEEE Press.

[24] A. De Luca, R. Salvaterra, A. Tiengo, D. D'Agostino, M.G. Watson, F. Haberl, J. Wilms (2016). Science with the EXTraS Project: Exploring the X-Ray Transient and Variable Sky. In The Universe of Digital Sky Surveys, 291-295. Springer, Cham.

[25] D. D'Agostino L. Roverelli, G. Zereik, G. La Rocca, A. De Luca, R. Salvaterra, A. Belfiore, G. Lisini, G. Novara, A. Tiengo (2017). A Science Gateway for Exploring the X-Ray Transient and Variable Sky Using EGI Federated Cloud. Submitted for publication to Future Generation Computing Systems.

[26] Q. Harpham, O. Gimeno, A. Parodi, D. DAgostino, (2017). A stakeholder consultation into hydro-meteorological e-science environments. Earth Science Informatics, 10(2), 219-234.

[27] A. Parodi, D. Kranzlmueller, A. Clematis, E. Danovaro, A. Galizia, L. Garrote, et al. (2017). DRIHM(2US): an e-Science environment for hydro-meteorological research on high impact weather events. Bulletin of the American Meteorological Society, in press. DOI: 10.1175/BAMS-D-16-0279.1

[28] A. De Luca, R. Salvaterra, A. Tiengo, D. D'Agostino, M.G. Watson, F. Haberl, J. Wilms (2017) EXTraS: Exploring the X-Ray Transient and Variable Sky. The X-ray Universe 2017 Symposium abstract book, 197.

[29] E. Danovaro, L. Roverelli, G. Zereik, A. Galizia, D. D'Agostino, A. Quarati, A. Clematis, et al. (2014). Setting up an hydro-meteo experiment in minutes: the DRIHM e-Infrastructure for HM research. In IEEE 10th International Conference on e-Science, 1, 47-54. IEEE.

[30] J. Michalakes, J. Hacker, R. Loft, M.O. McCracken, A. Snavely, et al. WRF nature run. (2008). WRF nature run. In Journal of Physics: Conference Series, 125(1), 012022. IOP Publishing.

[31] J. Ploski, G. Scherp, T.I. Petroliagis, O. Büchner, W. Hasselbring (2009). Grid-based deployment and performance measurement of the Weather Research & Forecasting model. Future Generation Computer Systems, 25(3), 346-350.

[32] V. Fernández-Quiruelas, J. Fernández, A.S. Cofiño, C. Blanco, M. García-Díez, M. Magariño, L. Fita, J.M. Gutiérrez (2015) WRF4G: WRF experiment management made simple, Geosci. Model Dev. Discuss., in review. Doi: 10.5194/gmdd-8-6551-2015